

Remarks

In the final Office Action, claims 1-4, 9-13, 18 & 19 were rejected under 35 U.S.C. §102(e) as being anticipated by Wollrath et al. (U.S. Patent No. 6,487,607 B1; hereinafter Wollrath); claims 1, 10 & 19 were rejected under 35 U.S.C. §103(a) as being unpatentable over Bahrs et al. (U.S. Patent No. 6,292,933 B1; hereinafter Bahrs) in view of Jones et al. (U.S. Patent No. 6,629,154 B1; hereinafter Jones); and claims 5-8, 14-17 & 20 were rejected under 35 U.S.C. §103(a) as being unpatentable over Wollrath in view of Skinner et al. (U.S. Patent No. 6,721,740 B1; hereinafter Skinner). These rejections are respectfully traversed and reconsideration thereof is requested for the reasons set forth below.

Applicants' recited invention is a protocol for transferring *executable program code* between computer processes, along with the data to be input to the executable program code. For example, claim 1 recites:

- providing an object which provides a hashtable, the hashtable having at least one set of elements, *one element of the at least one set of elements comprising executable program code*,
- wherein the executable program code includes logic *which employs as data input at least one other element of the hashtable*,
- transferring the object from a sender computer process to a receiver computer process, retrieving the executable program code from the hashtable and invoking the executable program code with at least one other element of the hashtable as data input thereto.

Advantageously, within Applicants' recited hashtable, both the executable program code and the data input to the logic of the executable program code are provided as separate elements. No similar protocol is taught or suggested by Wollrath, Bahrs, Jones or Skinner, either alone or in combination.

Initially, Applicants respectfully submit that the phrase *executable program code* is a well-known term of art meaning program code in an executable format, such as program code that has been converted into an executable file by a compiler. *Executable program code is program code that can be executed or run on a computer*, and is distinct from parameters or data input employed by the executable program code when executing. The discussion below of

Applicants' novel protocol relies on these definitions of the phrases employed in the recited invention. The rejections to the pending claims are addressed separately below in the order raised in the Office Action.

In rejecting claims 1-4, 9-13, 18 & 19 as anticipated by Wollrath, the Office Action relies upon steps 702-710 of FIG. 7, as well as the discussion thereof at column 10, line 41- column 11, line 14. In describing FIG. 7, these lines teach:

FIG. 7 is a flow chart of a process 700 for RMI using generic code not specific to any particular type of remote objects on which the method is invoked. An application in client machine 601 invokes a method on a stub that is an instance of the generic proxy class (step 701). The generic proxy instance's "invoke" method is called with the method object and parameter list bundled in an object array (step 702). The generic proxy transmits an identifier, which may include an object id, method hash and marshals parameter to a server machine (step 703).

Further details on the use of a method hash are disclosed in U.S. patent application entitled "Method and System for Deterministic Hashes to Identify Remote Methods," which was previously incorporated herein by reference. Marshalling involves constructing an object from a byte stream including code or a reference to code for use in the construction. Marshalling and unmarshalling are explained in U.S. Patent application Ser. No. 08/950,756, filed on Oct. 15, 1997, and entitled "Deferred Reconstruction of Objects and Remote Loading in a Distributed System," now allowed, which is incorporated herein by reference. (emphases added.)

Server machine 606 reads the object id and method hash (step 704), and it looks up the method object for the call using the method hash (step 705). Server machine 606 unmarshals the parameters for the operation given the types of the parameters specified in the method object (step 706). Step 706 may involve building a method table, which compiles values for particular methods and is initialized when a remove object is created and exported. The generic code creates a correspondence between a method hash and a particular method object. Thus, by using the hashes in the method table, different skeletons typed to different types of objects is not required for method invocation to correspond method hash to method object.

Server machine 606 invokes the method on the remote object implementation (step 707), and it marshals the return result(s) to the caller using the return type information in the method object (step 708). The generic proxy unmarshals the return result(s) using the return type information in the method object (step 709) and returns result(s) to the caller, client machine 601 (step 710).

The above-italicized Application incorporated by reference in Wollrath issued as U.S. Patent No. 6,134,603. Additionally, a continuation thereof issued as U.S. Patent No. 6,629,154 B1. The Abstracts of these two incorporated patents provide further details on the use of the method hash in the protocol being employed by Wollrath. These Abstracts teach:

A method and system is provided to uniquely identify a remote method to invoke on a server using a hash value computed from the method signature sent from the client to the server with the call request. When a client wishes to invoke a remote method located on a server, the client sends a hash value identifying the remote method to the server in the "remote method invocation" (RMI) call. In one implementation, this hash value is created by applying a hash function to the method string name and the parameter type list and possibly the return type. When the server receives the RMI call, the server identifies which method is being called using the received hash value. The server maintains a mapping of hash values to their associated remote methods located on the server and references the correct method using the hash value. Additionally, in one implementation, the server creates the mapping table dynamically when a remote object is created. The server identifies the methods implemented by the object and creates hash values for each method. These hash values are stored in a mapping table which is used to reference the remote methods.

As expressly taught in the Abstract for these two patents, Wollrath provides a method and system to *uniquely identify a remote method to invoke on a server using a hash value computed from the method signature sent from the client to the server with the call request*. When a client wishes to invoke a remote method located on a server, the client sends a hash value identifying the remote method to the server in the "remote method invocation" (RMI) call. In one implementation, the hash value is created by applying a hash function to the method string *name* and the parameter type list and possibly the return type.

As clearly taught by the above-cited lines of Wollrath, the Wollrath protocol applies a hash function to the *method name* and parameter type list. The method name is distinct from the executable program code itself, and is merely a name for code. In Applicants' invention, the executable program code itself is part of the hashtable being transferred from the sender computer process to the receiver computer process. Thus, the cited lines of Wollrath actually teach away from Applicants' recited invention. For at least this reason, Applicants respectfully submit that there is no anticipation of their independent claims based upon Wollrath.

Additionally, Applicants respectfully submit that the RMI protocols is a well-established protocol. *In RMI, it is assumed that the method to be invoked is already resident at the remote process, i.e., the server.* Further, in RMI, the sender and receiver have a knowledge agreement as to the methods to be invoked via a registering process. This is in contrast with Applicants' recited invention, wherein the *executable program code* is essentially "pushed" as part of the hashtable from the sender computer process to the receiver computer process. No similar teaching is provided by Wollrath, or the other art-of-record.

In summary, Wollrath expressly teaches at column 10, lines 51-54 that details of the method hash are provided in the incorporated patents. These incorporated patents expressly teach the creation of a hash from the method *signature* which is sent from the client to the server. There is no teaching or suggestion in Wollrath of transferring within an object a hashtable *which includes executable program* per se. Rather, in Wollrath, RMI protocol is employed which again assumes the existence on the server (i.e., the remote process) of the program code (i.e., method) to be invoked. Since Wollrath expressly teaches the use of RMI, Wollrath does not anticipate Applicants' invention as set forth in the pending independent claims. Each of Applicants' independent claims recite that the object being transferred from the sender computer process to the receiver computer process includes a hashtable, one element of which is *executable program code*, with one other element of the hashtable being data input to the executable program code. Thus, Applicants respectfully request withdrawal of the anticipation rejection to their independent claims based upon Wollrath.

Independent claims 1, 10 & 19 were also rejected as obvious over Bahrs in view of Jones. Reconsideration and withdrawal of this rejection are respectfully requested.

Bahrs describes a method and apparatus for serializing data. A serializer receives a data element for serialization, and the data element includes a class-name string. Responsive to receiving the data element, the serializer replaces the class-name string with a code having a smaller size than the class-name string to form a modified data element. Responsive to forming the modified data element, the serializer serializes the modified data element. The serialized data element is transmitted and deserialized by a deserializer which replaces the indicator with the class name. (See Abstract of Bahrs.)

Initially, Applicants respectfully submit that Bahrs is not relevant to the invention recited herein. Bahrs describes a serializer for serializing *a data element*. Bahrs teaches that the data element includes *a class-name string*. In Bahrs, the class-name string is replaced with a “code” having a smaller size than the class-name string to form a modified data element. Clearly, the “code” referred to in Bahrs is an encoded or shortened version of the class-name string. Bahrs expressly teaches this in the Abstract where it is stated that the “code” has a smaller size than the class-name string and forms *a modified data element*. Thus, there is no teaching, suggestion or implication in Bahrs of an executable program code being included with a hashtable, as recited in Applicants’ independent claims. To the extent the Office Action asserts otherwise, it is believed a mischaracterization of the teachings of Bahrs.

In the Office Action, column 4, lines 53-63, column 60, lines 16-35 & column 66, lines 34-59 are cited. These cited lines of Bahrs simply provide additional details on the above-noted functionality summarized in the Abstract. In Bahrs, a class *name* is replaced with an indicator having a smaller size than the class name to form a modified *data element*. Claim 2 at column 66 of Bahrs describes the step of replacing the class-name string with the indicator of smaller size. This step includes hashing the class name to create a hash code and replacing the class name with a hash code. The phrase “hash code” again refers to an encoded version of the class name, and not to executable program code as the phrase as understood by one skilled in the art. Since there is no executable program code in Bahrs comprising an element of a hashtable, it is respectfully submitted that Applicants’ invention patentably distinguishes over the teachings of Bahrs and Jones, alone or in combination.

Jones describes a method and system for deterministic hashes to identify remote methods. When a client wishes to invoke a remote method located on a server, the client sends a hash value identifying the remote method to the server in the “remote method invocation” (RMI) called. In one implementation, this hash value is created by applying a hash function to a method-string *name* and the parameter type list and possibly return type. When the server receives the RMI call, the server identifies which method is being called, using the received hash value. (See Abstract of Jones.)

Since Jones expressly teaches an RMI-based protocol, Applicants respectfully submit that Jones does not teach the above-noted deficiencies of Bahrs when applied against the pending independent claims. Neither patent teaches Applicants' recited protocol providing an object comprising a hashtable, wherein one element of the hashtable is *executable program code*, and another element of the hashtable is data input which is employed by the executable program code. Using Applicants' recited object, the executable program code itself, and the data input to the code, are both pushed or transferred from a sender computer process to a receiver computer process. In both Bahrs and Jones, a remote method invocation call is employed wherein an executable program code resident on the receiving computer system is identified by the data sending computer system, but not transferred from the data sending computer system, let alone in a hashtable as recited in Applicants' protocol. Combining Jones with Bahrs combines one RMI protocol with another RMI protocol. The resultant combination is still an RMI protocol, which is distinct from Applicants' recited processing wherein the *executable program code* itself is included as part of the hashtable forwarded from the sender computer process to the receiver computer process, along with the input data employed by the executable program code. No similar processing is taught or suggested by the combined RMI protocols of Bahrs and Jones. Thus, reconsideration and withdrawal of the rejection based thereon are respectfully requested.

For at least the above-noted reasons, Applicants respectfully submit that the pending independent claims patentably distinguish over the applied art. The dependent claims are believed allowable for the same reasons as the independent claims, as well as for their own additional characterizations.

For example, claims 4, 13 & 20 further characterize the respective independent claims by reciting protocol which includes adding data to the hashtable at the receiver computer process. The added data at the receiver computer process is indicated to be relevant to the executable program code and is *added prior to invoking of the executable program code using as data input only the hashtable*. Applicants respectfully submit that the careful reading of Wollrath fails to teach or suggest this protocol. Cited against these claims is step 706, as well as column 10, line 64 – column 11, line 7 of Wollrath. Step 706 in FIG. 7 of Wollrath states: "Server unmarshalls

the parameters for the operation given the types of parameter specified in the method object.”
The cited lines of column 10 & 11 state:

“...Server machine 606 unmarshalls the parameters for the operation given the types of the parameters specified in the method object (step 706). Step 706 may involve building a method table, which compiles values for particular methods and is initialized when a remote object is created and exported. The generic code creates a correspondence between a method hash and a particular method object. Thus, by using the hashes in the method table, different skeletons typed to different types of objects is not required for method invocation correspond method hash to method object.”

Applicants respectfully submit that the above-noted teachings from Wollrath are not relevant to the recited protocol. In Applicants’ dependent claims 4, 13 & 20 the recited invention includes *adding data to the hashtable at the receiver computer process*. There is no data taught in step 706 of FIG. 7 or the accompanying discussion in columns 10 & 11 being added to a hashtable received at a receiver computer process. As such, these claims are believed patentable over the applied art.

Still further, dependent claims 4, 13 & 20 recite that the executable program code *uses as data input only the hashtable*. Thus, the data input recited in claims 4, 13 & 20 is data provided by the sender computer process placed into the hashtable before sending of the object and data placed into the hashtable by the receiver computer process after receipt of object. Again, there is no similar teaching or suggestion in the applied art. For at least the above-noted reasons, Applicants respectfully request reconsideration and allowance of these claims.

Dependent claims 7 & 16 further indicate that the first hashtable received from the first sender computer process and the second hashtable received from the second sender computer process are merged at the receiver computer process into a common hashtable. In rejecting this subject matter, the Office Action relies upon Wollrath and Skinner. In particular, lines 34-51 of Skinner. These lines teach:

“Serialization provides a useful mechanism for object persistence and transmission. With respect to persistence, a serialized object may be stored in memory any length of time and regenerated with the same state the object had at the time it was serialized. With respect to object transmission, a serialized object may be transmitted between remote clients and servers as a data streams or packets in accordance with known communication protocols, such as the protocol implemented by the Remote Method Invocator (RMI) API. Also, serialized objects may be written to a shared portion of memory by one application and read out of the shared memory by another application. Client-side and server-side communication management components 305A and 305B implement methods to perform serialization and deserialization functions. Data transmission and reception of serialized objects, may be implemented using any known communication protocol as described above.”

Applicants respectfully submit that the above-noted teachings from the Skinner patent are simply not relevant to the recited functionality. In Applicants’ recited invention, the hashtable includes executable program code to be invoked. In Skinner, objects and method calls are transmitted between a client and application server. Objects configured with metadata can be serialized, i.e. broken down into a set of data bytes and later reconstituted (i.e., deserialized by the same or different application) to generate a copy of the original object. It is respectfully submitted that this breaking down of objects configured with metadata refers to a data stream, not to a hashtable containing executable program code. There is no discussion in Skinner that the objects configured with metadata include executable program code within a hashtable. As such, Applicants respectfully submit that claims 7 & 16 patentably distinguish over Wollrath and Skinner taken together. Further, there is no teaching or suggestion in Skinner of iterating through the common hashtable for executable program code to be invoked *using the common hashtable as the only data input thereto*. For at least these reasons, Applicants request reconsideration allowance of dependent claims 7 & 16.

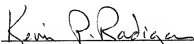
Dependent claims 8 & 17 are believed allowable for the same reasons noted above respect to claims 7 & 16 as well as those reasons stated above respect to dependent claims 4, 13 & 20. These dependent claims further recite that the receiver computer process adds data to the common hashtable prior to invoking the executable program code using as data input only the common hashtable. In the Office Action, Skinner is cited against the subject matter of claims 8

& 17. In particular, column 16, lines 43-51 discuss serialized objects which may be written to a shared portion of memory in one application and read out of the shared memory by another application. Client side and server side communication management components implement to perform serialization and deserialization functions. These teachings are simply not relevant to the functionality recited in Applicants' dependent claims 8 & 17. There is no teaching or suggestion in Skinner (or in Wollrath) of a receiver computer process *adding data to a hashtable* which is relevant to the received executable program code within the hashtable and which is added prior to invoking the executable program code *using as data input only the hashtable*. As such, Applicants respectfully submit that these claims patentably distinguish over the applied art.

All claims are believed to be in condition for allowance and such action is respectfully requested.

Should any issue remain unresolved, Applicants' undersigned representative requests a telephone interview with the Examiner to further discuss the matter in the hope of advancing prosecution of the subject application.

Respectfully submitted,


Kevin P. Radigan, Esq.
Attorney for Applicants
Reg. No.: 31,789

Dated: December 11, 2007

HESLIN ROTHENBERG FARLEY & MESITI P.C.
5 Columbia Circle
Albany, New York 12203
Telephone: (518) 452-5600
Facsimile: (518) 452-5579